Indices

Thomas Schwarz, SJ

Indices

- For a DBMS administrator:
  - Important to make common queries fast
  - E.g.: Lookup by name can be a frequent occurrence
- To speed up these queries, we use indices
  - "Indexes" in SQL, which treats it as an English word
- Index is a data structure that implements a generalized dictionary or key-value store
  - Given a key, find all records with that value
  - Unlike a dictionary / key-value store: keys can have multiple values

Indices

- Indices come at a cost
  - Need to be maintained at all updates, insertions, deletes

Indices

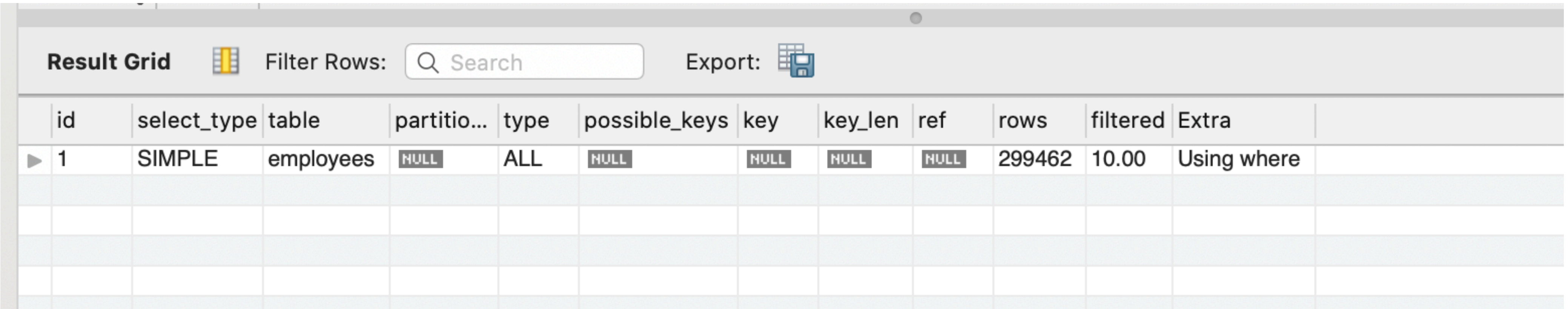
- Why can indices make a difference?
  - Usually, tables are stored in pages of SSD or blocks of HDD
  - Fetching a page to look up it costs time
    - SSD response time: ~10 μsec
    - HDD response time: ~ 5 msec
- An index can minimize the amount of data that needs to be fetched

MySQL Example

- To see how indices work, we can use the EXPLAIN statement in MySQL
- We use the employees database
- We look at a simple SELECT WHERE query
- We create an index
- We look at the same simple SELECT WHERE query

MySQL Example

```
EXPLAIN SELECT *
FROM employees
WHERE last_name = 'Rosis';
```



id	select_type	table	partitio...	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	employees	NULL	ALL	NULL	NULL	NULL	NULL	299462	10.00	Using where

MySQL Example

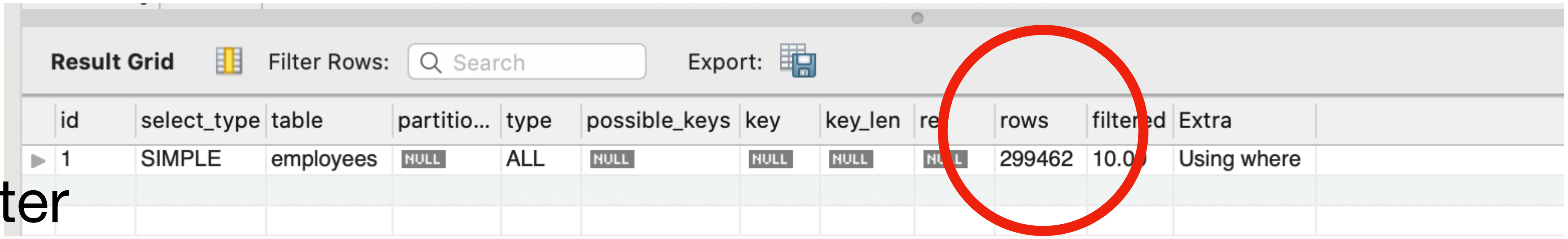
- Create an Index on last_name

```
CREATE INDEX iLastName ON employees(last_name);
```

Example MySQL

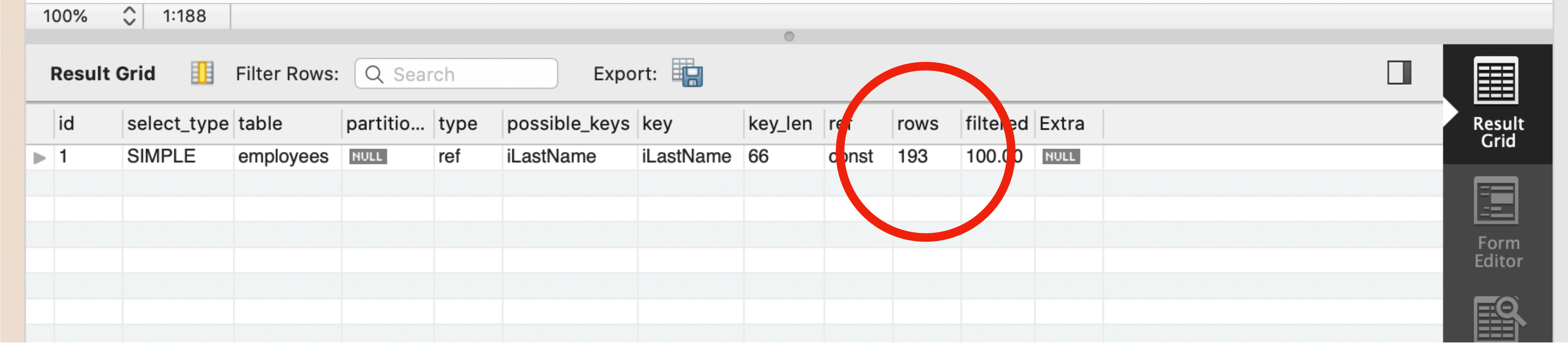
```
EXPLAIN SELECT *
FROM employees
WHERE last_name = 'Rosis';
```

Before



id	select_type	table	partitio...	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	employees	NULL	ALL	NULL	NULL	NULL	NULL	299462	10.00	Using where

After



id	select_type	table	partitio...	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	employees	NULL	ref	iLastName	iLastName	66	const	193	100.00	NULL

Example MySQL

- Without the index, the query looked at all the rows
- With the index, the query located just a few hundred rows

- We can use the SHOW INDICES FROM tablename to display all indices

Example MySQL

- There are three indices in my version

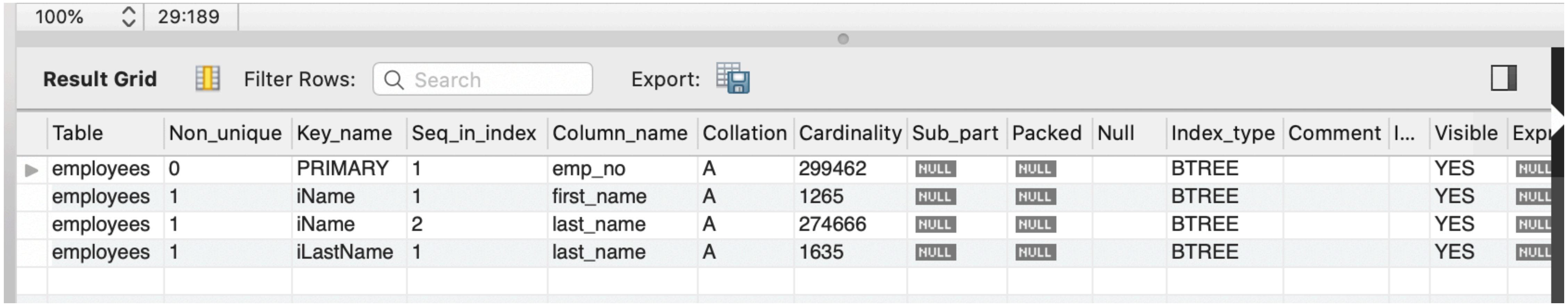


Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	I...	Visible	Exp
employees	0	PRIMARY	1	emp_no	A	299462	NULL	NULL		BTREE			YES	NULL
employees	1	iName	1	first_name	A	1265	NULL	NULL		BTREE			YES	NULL
employees	1	iName	2	last_name	A	274666	NULL	NULL		BTREE			YES	NULL
employees	1	iLastName	1	last_name	A	1635	NULL	NULL		BTREE			YES	NULL

- One created because emp_no is a primary key
- One called iName, and the one we just created: iLastName

Indices

- Some indices are created automatically
  - DBMS needs them to enforce constraints
    - Primary key
    - Foreign key

MySQL Example

- Example: dept_emp in employees has a primary key and a foreign key restraint.
  - Both result in an index
    - Primary key is two attributes
    - Foreign key is one attribute

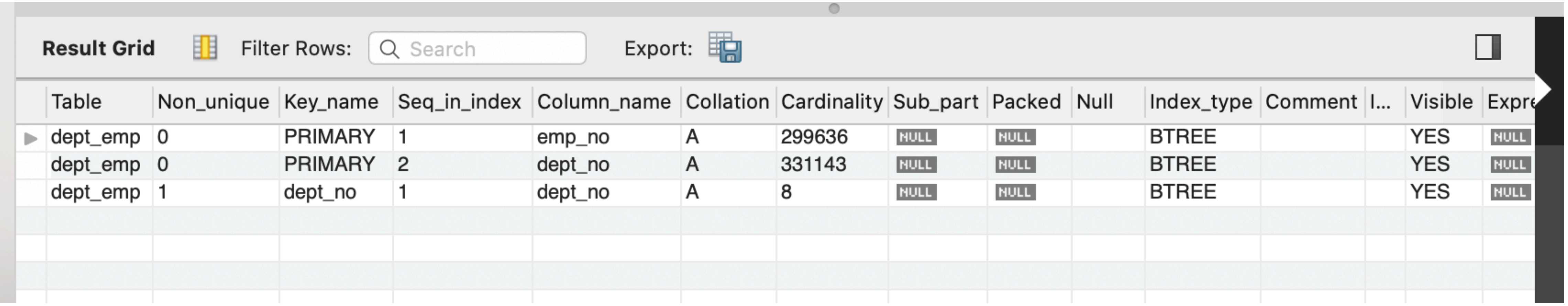


Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	I...	Visible	Expre
dept_emp	0	PRIMARY	1	emp_no	A	299636	NULL	NULL		BTREE			YES	NULL
dept_emp	0	PRIMARY	2	dept_no	A	331143	NULL	NULL		BTREE			YES	NULL
dept_emp	1	dept_no	1	dept_no	A	8	NULL	NULL		BTREE			YES	NULL

Indices

- Indices where standardized in SQL-99
  - Even though most commercial database products had them
  - Typical syntax
    - `CREATE INDEX indexname ON tablename(listofcolumns)`
  - If you specify more than one column:
    - Only speeds up searches that specify values for all of these columns
      - E.g.: In the MySQL example, the index on first and last name did not speed up a query for last name only

Indices

- During the table creation, you can just specify the indices you want

```
CREATE TABLE t(
   c1 INT PRIMARY KEY,
   c2 INT NOT NULL,
   c3 INT NOT NULL,
   c4 VARCHAR(10),
   INDEX (c2,c3)
);
```

- You can also drop an index

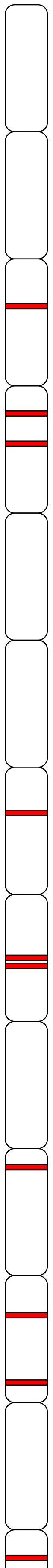
```
DROP INDEX iName;
```

Indices

- Effectiveness of indices
    - Cost of indices: More work for updates, inserts, deletes
    - Benefits of indices: Can reduce the number of pages fetched
        - Looking at one record in a page takes almost as long as looking at all records in a page
        - Effect depends on:
            - What is your storage type
                - Hint: You can spend money on Intel Optane storage to speed it up
            - How clustered the records are that are indexed

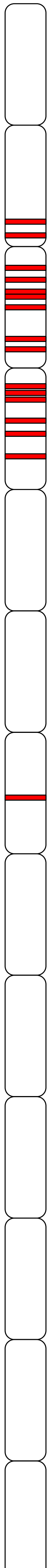
Indices

- Records with indexed value can be scattered over storage
    - Use of the index only reduces number of pages by half



Indices

- Relevant records are clustered
  - Need only retrieve a few pages



Indices

- Clustering depends on the intrinsic design of a database management system
- However, if we only look for few records with a given value, then indexing is bound to be effective

Indices

- Example:
  - `starsIn(movieTitle, movieYear, starName)`
  - Assume we have a frequent query

    -
      ```
      SELECT movieTitle, movieYear
      FROM starsIn
      WHERE starName = s;
      ```
  - Should we build an index on starName?

Indices

- Example (cont):
    - Each year, there are about 750 movies to put into a database
    - Assume we have a database starting at 1950
        - That would give us about 50,000 movies
            - But there were more movies earlier
                - So let's say 100,000 movies in the database

Indices

- Example (cont):
  - On average, we might have three or four stars per movie in our starsIn database
  - Table should have 400,000 entries
  - Each entry has about 50 B (big assumption)
  - So, total size of table is 2,000,000 B = 2 MB
  - Blocks have size 4KB, so about 500 blocks

Indices

- Example (cont):
    - John Wayne has about 150 movies with credits
    - Carrie Fisher has about 30 movies with credits
    - Average is probably closer to the lower range: 30 movies per star on average
- Without index: Need to fetch 500 pages
- With index in the worst case:
    - Need to fetch 30 pages
- Index fetches ~20 times less pages, so let's go for it if the query is frequent

Indices

- Example:
  - What about the opposite query

    -
      ```
      SELECT starName
      FROM starsIN
      WHERE movieTitle = 'Rio Hondo'
         and movieYear = 1959;
      ```
  - Even better, about four entries per title / year
    - Fetch about four blocks out of 500
  - Index speeds up fetching by a factor of 100
    - Close to actual wall-clock timing update

Indices

- Example:
    - However, if these queries are extremely rare, then the gain is not realized
    - Cost of maintaining indices depends on the number of entries:
        - In our case, about 750 movies are entered into the database
        - About 3000 updates per year
            - That is not a lot

Indices

- Example:
  - Transactions at an e-auction house
    - Any bid, any offer entered into a database
    - Updates almost as frequents as queries
    - Need to be very careful about the costs of indexing

Materialized Views

- Views are virtual
    - Created whenever they are accessed
    - But views can be heavily used
        - Views are used to:
            - Easier query logic because the definition of the view encompasses the difficulties
                - E.g. a view that uses a join of many tables
            - Security: Restrict access to tables, but give access to views
            - Enforce business rules: What is "active", what is "popular"

Materialized Views

- Virtual views that are heavily used means
  - running a query against a view
  - running a query to recreate the view
- Materialized views store the view in a derived table
  - Not all DBMS support materialized views
  - Some give it a different name
- Typical command:

```
CREATE MATERIALIZED VIEW movieProd AS
   SELECT title, year, name
   FROM movies, movieExec
   WHERE procuderC# = cert#
```

Materialized Views

- Materialized views need to be maintained
    - Some updates / inserts / deletes to movieExec and movies need to be intercepted
    - The changes to the materialized view are incremental

Materialized Views in MySQL

- They do not exists as materialized views
- But we can work around it:
    - Materialized views are tables that are modified by modifications to the base tables
    - Can use triggers to intercept modifications of the base tables in order to update the materialized view